

Computer Graphics

3rd class

أستاذ الماده: م.م. ياسر منذر

Computer Graphics

1.1 The term computer graphics involves using a computer to create and hold pictorial information and also to adept and manipulate the display in different ways.

A computer is capable of sending its output to wide variety of devices, many of which are designed for special purposes. We will concern ourselves with devices that capable of producing graphical output on the computer's display.

1.2 The origins of computer graphics

In 1950 the first computer driven display, attached to MIT's computer, was used to generate simple pictures. This display used a Cathode-Ray tube (CRT). Interactive computer graphics made progress and the term computer graphics was first used in 1960.

1.3 Interactive Computer Graphics

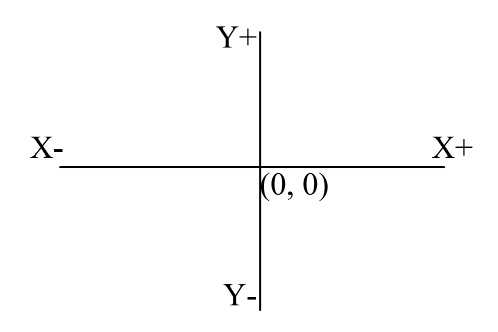
It involves two way communication between computer and user. The computer up on receiving signals from the input device, can modify the displayed picture appropriately.

The main reason for the effectiveness of interactive computer graphics in many applications is the speed with which the user of the computer can assimilate the display information.

1.4 Cartesian coordinate system

A coordinate system provide a framework for translating geometric ideas into numerical expressions.

In a two-dimensional plane, we pick any point and single it out as a reference point called the origin. Through the origin we construct two perpendicular number lines called axes. These are labeled the X axis and the Y axis. Any point in two dimensions in this X-Y plane can be specified by a pair of numbers, the first number is for the X axis, and the second number is for the Y axis.



1.5 How the Interactive Graphics display works

The modern graphics display is extremely simple in construction. It consists of three components:

1- A digital memory, or frame buffer, in which the displayed image is stored as a matrix of intensity values.
2- A monitor
3- A display controller, which is a simple interface that passes the contents of the frame buffer to the monitor.

Inside the frame buffer the image is stored as a pattern of binary digital numbers, which represent a rectangular array of picture elements, or pixel.

The pixel is the smallest addressable screen element. In the simplest case where we wish to store only black and white images, we can represent black pixels by 0's in the frame buffer and white pixels by 1's. The display controller simply reads each successive byte of data from the frame buffer and convert each 0 and 1 to the corresponding video signal. This signal is then fed to the monitor. If we wish to change the displayed picture all we need to do is to change of modify the frame buffer contents to represent the new pattern of pixels.

1.6 Graphical user interface

The aspects of the user interface of a program are the parts of the program that link the user to the computer and enable him to control it.

A good user interface makes the program not only easy to use and learn but also easier to operate and more efficient and vis versa.

1.7 Graphics Devices

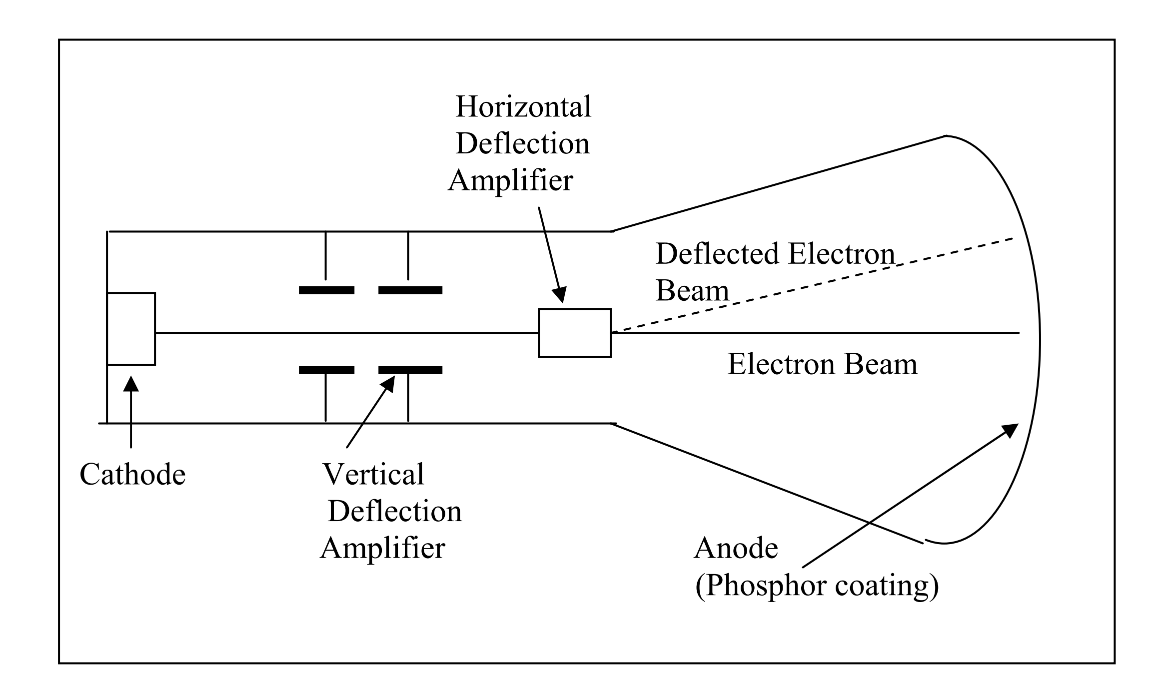
Typical examples are plotters, laser printer plotters, films, storage tube and raster scan cathode ray tube (CRT) display. Because the large majority of computer graphics systems utilize some type of CRT display and because most of the fundamental display concepts are embedded in CRT display technology, we will limit our concern to CRT display.

The three most common types of CRT display technologies are:

1- Direct view storage tube display
2- Calligraphic refresh display
3- Raster scan refresh display

Before discuses the CRT display, we must understand the CRT basics.

Cathode Ray Tube basics :
The CRT used in video monitor is shown in the figure below



A cathode (negatively charged) is heated until electron “boil” off in a diverging cloud (electron repel each other because they have the same charge). These electrons are attracted to a highly charged positive anode.
This is the phosphor coating on the inside of the face of the large end of the CRT. If allowed to continue uninterrupted, the electrons would simply flood the entire face of the CRT with a bright glow.

However, the cloud of electrons is focuses into a narrow, precisely collimated beam with an electron lens. At this point the focused electron beam produces a single bright spot at the center of the CRT. The electron beam is deflected or positioned to the left or right of the center and/or above or below the center by means of horizontal and vertical deflection amplifiers.

1.8 Type of graphics devices

1- Storage Tube Graphics Displays

This storage tube display, also called a bistable storage tube, can be considered a CRT with a long persistence phosphor. A line or character will remain visible up to a hour until erased. To draw a line on the display the intensity of the electron beam is increased sufficiently to cause the phosphor to assume its permanent bright “storage”. The display is erased by flooding the entire tube with a specific voltage which causes the phosphor to assume its dark state.

Features of storage tube graphics display

1- Storage tube display is flicker free
2- Capable of displaying an unlimited number of vectors
3- Resolution is typically 1024 X 1024 addressable points on an 8 X 8 inch square or 4096 X 4096 on either 14 X 14 or an 18 X 18 inch square.
4- Display of dynamic motion or animation is not possible.
5- A storage tube display is a line drawing or random scan display. This means that a line (vector) can be drawn directly from any addressable point to any other addressable point.
   This device plots continuous lines and curves rather than separate pixels.
6- A storage tube display is easier to program than a calligraphic or raster scan refresh display.
7- The level of interactivity is lower than with either a refresh or raster scan display.

2- The Calligraphic Refresh graphics display

A Calligraphic (line drawing or vector) refresh CRT display uses a very short persistence phosphor. These displays are frequently called random scan display. Because of the short persistence of the phosphor, the picture painted on the CRT must be repainted or refreshed many times each second. The minimum refresh rate is at least 30 times each second. Refresh rates much lower than 30 times each second result in a flickering image.

The basic calligraphic refresh display requires two elements in addition to the CRT. These are the display buffer and the display

controller. The display buffer is contiguous memory containing all the information required to draw the picture on the CRT. The display controller's function is to repeatedly cycle through this information at the refresh rate. Two factors which limit the complexity (number of vectors displayed) of the picture are the size of the display buffer and the speed of the display controller. A further limitation is the speed at which picture information can be processed.

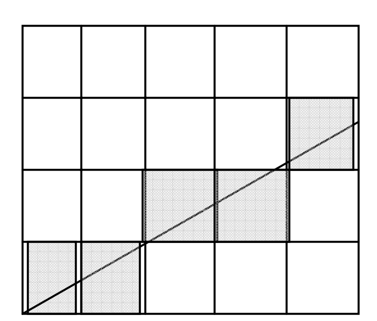
Features of calligraphic refresh displays

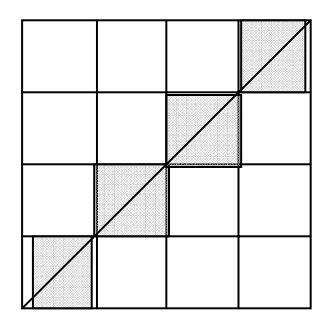
1- It is a vector graphics display
2- Resolution is the same as storage tube display
3- Employee the concept of picture segmentation that support the interactive graphics programs

3- Raster refresh graphics display

A raster CRT graphics devices can be considered a matrix of discrete cells each of which can be made bright. Thus it is a point plotting devices.

It is not possible except in special cases to directly draw a straight line from one addressable point, or pixel in the matrix to another addressable point, or pixel. The line can only approximated by a series of dots (pixels) close to the path of the line.





Only in the special cases of completely horizontal, vertical or 45 degree lines will a straight line result. All other lines will appear as a series of stair steps. *This is called aliasing*.

The most common method of implementing a raster CRT graphics device utilize a frame buffer. A frame buffer is a large, contiguous piece of computer memory. As a minimum there is one memory bit for each location or pixel in the raster. This amount of memory is called a bit plane.

A 512 X 512 element square raster requires 2^18 memory bits in a single bit plane. The picture is built up in the frame buffer 1 bit at a time.

Since a memory bit has only two states (0 or 1), a single bit plane yields a black and white display. Color or gray levels can be incorporated into a frame buffer raster graphics device by using additional bit planes.

The capacity of the frame buffer depends on the number of bits representing each pixel, on the number of pixels per scan line and on the number of the scan lines.

The CRT must be refreshed by repeatedly passing to it the image to be displayed. The image must be transmitted to the display point by point. Unless the entire image can be transmitted at least 25 times a second, it will begin to flicker.

1.9 Pixels and Frame buffer

We can not represent an infinite number of points (pixels) on a computer, just as we can not do that with numbers. The machine is finite, and we are limited to a finite number of points making up each line. The maximum number of points (pixels) which a line can have is a measure of the resolution of the display device. The greater the number of points, the higher the resolution.

Resolution is the number of visibly distinct dots that can be displayed in a given area of the screen

Computer Graphics

Line and Circle

1-Line Drawing algorithms

1-1 : A straight line may be defined by two endpoints and an equation

If the two endpoints used to specify a line are (X1,Y1) and (X2,Y2) , then the equation of the line is used to describe the X , Y coordinates of all the pointes that lie between these two endpoints

The equation of the straight line is :

$$Y = M * X + B$$

Where (M) is the slope of the line ➔ $M = \frac{\Delta Y}{\Delta X}$

and (B) is the Y intercept.

The Y values of the pointes of the line can be calculated using the above equation, by incrementing X values from X1 to X2 and substitute it in the line equation.

Note : The slope between any point (X,Y) on the line and (X1,Y1) is the same as the slope between (X2,Y2) and (X1,X2)

$$\frac{Y-Y1}{X-X1} = \frac{Y2-Y1}{X2-X1}$$

1-2 : DDA (Digital Differential Analyzer) algorithm

The DDA algorithm generates lines from their differential equations.

We calculate the length of the line in the X direction (number of pointes) by the equation :

ABS (X2-X1)

and calculate the length of the line in the Y direction (number of pointes) by the Equation :

ABS (Y2-Y1)

Where *ABS* is a function takes the positive of the arguments.

The Length estimates is equal to the larger of the magnitudes of the above two equations.

The increment steps (dX and dY) are used to increment the X and Y coordinates for the next pointes to be plotted

$$dX = \frac{X2 - X1}{\text{Larger Length}} \qquad dY = \frac{Y2 - Y1}{\text{Larger Length}}$$

Algorithm DDA

Start

```
If ABS( X2-X1) > ABS (Y2-Y1) Then
                              Length=ABS (X2-X1)
                         Else
                              Length=ABS (Y2-Y1)

dX = (X2-X1) / Length
dY = (Y2-Y1) / Length

X=X1+ 0.5 * Sign(ΔX)
Y=Y1+ 0.5 * Sign(ΔY)

For I=1 to Length
  Begin
      Plot(Int(X) , Int (Y))
      X=X+dX
      Y=Y+dY
  End
```

Finish

<u>Note :</u>

1- Sign function returns : -1 if its argument is < 0
   : 0 if its arguments is = 0
   : +1 if its arguments is > 0

   Ex. Sign(-10) = -1   Sign (5) = 1

   Using the Sign function makes the algorithm work in all quadrants.

2- Int function works as follow :
   Int (8.5) = 8
   Int (-8.5) = -9

Example 1 : Consider the line from (0,0) to (5,5)
Use DDA to rasterize the line.

Sol 1 :

X1=0 ; Y1=0 ; X2=5 ; Y2=5 ; Length=5
dX=1 ; dY=1 ; X=0.5 ; Y=0.5

I	Plot	X	Y
		0.5	0.5
1	(0,0)	1.5	1.5
2	(1,1)	2.5	2.5
3	(2,2)	3.5	3.5
4	(3,3)	4.5	4.5
5	(4,4)	5.5	5.5

Note : the integer part of X and Y are used in plotting the line.
This would normally have the effect of truncating rather then rounding so we initialize the DDA with the value 0.5 in each of the fractional parts to achieve true rounding. One advantage of this arrangement is that it allows us to detect changes in X and Y and hence to avoid plotting the same point twice

Example 2 : Consider the line from (0,0) to (-8,-4) ; evaluate the
DDA algorithm

Sol 2 :

X1=0 ; Y1=0 ; X2=-8 ; Y2=-4 ; Length =8
dX=-1 ; dY=-0.5 ; X=-0.5 ; Y=-0.5

i	plot	X	Y
		-0.5	-0.5
1	(-1,-1)	-1.5	-1
2	(-2,-1)	-2.5	-1.5
3	(-3,-2)	-3.5	-2
4	(-4,-2)	-4.5	-2.5
5	(-5,-3)	-5.5	-3
6	(-6,-3)	-6.5	-3.5
7	(-7,-4)	-7.5	-4
8	(-8,-4)	-8.5	-4.5

Features of DDA

1- The algorithm is orientation dependent
2- The end point accuracy deteriorates
3- The algorithm suffer from the fact that it must be performed using floating point arithmetic

1-3 : Bresenham's algorithm

Bresenham algorithm seeks to select the optimum raster locations to represent a straight line. To accomplish this the algorithm always increment by one unit in either X or Y depending on the slop of the line. The slop of the line represent the error between the location of the real line and the location of the drawn line.

The algorithm is cleverly constructed so that only the sign of this error need be examined.

Bresenham algorithm for the first octant : $\{ 0 \le \Delta Y \le \Delta X \}$

```
Start

 X=X1
 Y=Y1

 DX=X2-X1
 DY=Y2-Y1

 E= (DY / DX) – 0.5

 For I=1 to DX
 Begin
      Plot (X,Y)
      While  (E ≥ 0 )
                    Y=Y+1
                    E=E-1
      End While
      X=X+1
      E=E + ( DY / DX )
 End

 Finish
```

Example 3 : Consider the line from (0,0) to (5,5)
Rasterize the line with Bresenham algorithm

Sol 3 : X=0 ; Y=0 ; DX=5 ; DY=5 ; E=0.5

I	Plot	E	X	Y
		0.5	0	0
1	(0,0)	-0.5	0	1
		0.5	1	1
2	(1,1)	-0.5	1	2
		0.5	2	2
3	(2,2)	-0.5	2	3
		0.5	3	3
4	(3,3)	-0.5	3	4
		0.5	4	4
5	(4,4)	-0.5	4	5
		0.5	5	5

1-4 : Integer Bresenham algorithm

Bresenham algorithm requires the use of floating point arithmetic and division to calculate the slop of the line and to evaluate the error term. The speed of the algorithm can be increased by using integer arithmetic and eliminating the division.

Since only the sign of the error term is important, the simple transformation

$$\bar{E} = E * 2 * \Delta X$$

of the error term in the previous algorithm yields an integer algorithm. This allows the algorithm to be efficiently implemented in hardware.

So : $E=\{ (DY/DX) - 0.5 \} * 2\,\Delta X \rightarrow E = 2\,\Delta Y - \Delta X$

$E=\{ E-1 \} * 2\,\Delta X \rightarrow E = E - 2\,\Delta X$

$E=\{ E + (DY/DX) \} * 2\,\Delta X \rightarrow E = E + 2\,\Delta Y$

Bresenham's integer algorithm for the first octant { $0 \leq \Delta Y \leq \Delta X$ }
i.e. slop between zero and one

```
Start

X=X1
Y=Y1

dX=X2-X1
dY=Y2-Y1

E= 2 ΔY - ΔX

For I=1 to dX
Begin
    Plot (X,Y)

    While (E ≥ 0 )
    Begin
        Y=Y+1
        E= E - 2 ΔX
    End While

    X=X+1
    E = E+ 2 ΔY

Next I

Finish
```

1-5 : General Bresenham's algorithm

A full implementation of Bresenham algorithm requires modification for lying in the other octanats. These can easily be developed by considering quadrant in which the line lies and its slope. When the absolute magnitude of the slop of the line is > 1 , Y is incremented by one end Bresenham error is used to determine when to increment X.

Whether X or Y incremented by +(-) 1 depends on the quadrant.

General Bresenham's algorithm for all quadrants

```
X=X1
Y=Y1
dX=Abs (X2-X1)
dY=Abs(Y2-Y1)
S1=Sign (X2-X1)
S2=Sign (Y2-Y1)

If dY > dX Then
Begin
     T=dX  :  dX=dY  :  dY=T   :  Interchange=1
End
Else
   Interchange =0
End If
E= 2 dy - dx
For I=1 to dX
 Plot (X,Y)
 While ( E ≥ 0)
 Begin
     If  Interchange=1 Then  X=X + S1
                         Else  Y= Y + S2
     End If
    E= E - 2 dx
 End While
 If Interchange=1 Then  Y=Y + S2
                   Else  X=X + S1
End If
E = E+ 2 dy
Next I
Finish
```

Example 4 : Draw the line from (0,0) to (-8,-4) using General Bresenham algorithm

Sol 4 : X=0 ; Y=0 ; dX=8 ; dY=4 ; S1=-1 ; S2=-1

Because dX>dY then Interchange=0 ; E=0

I	Plot	E	X	Y
		0	0	0
1	(0,0)			
		-16	0	-1
		-8	-1	-1
2	(-1,-1)			
		0	-2	-1
3	(-2,-1)			
		-16	-2	-2
		-8	-3	-2
4	(-3,-2)			
		0	-4	-2
5	(-4,-2)			
		-16	-4	-3
		-8	-5	-3
6	(-5,-3)			
		0	-6	-3
7	(-6,-3)			
		-16	-6	-4
		8	-7	-4
8	(-7,-4)			
		0	-8	-4

2- Circle Drawing algorithms

Bresenham algorithm for circle drawing

In addition to drawing a straight line we need to drae a circle , ellipse, etc.

To begin with circle drawing, note that only one octant of the circle need to be generated. The other parts can be obtained by successive reflections. To drive Bresenham circle generation algorithm , consider the first quadrant of the origin centered circle.

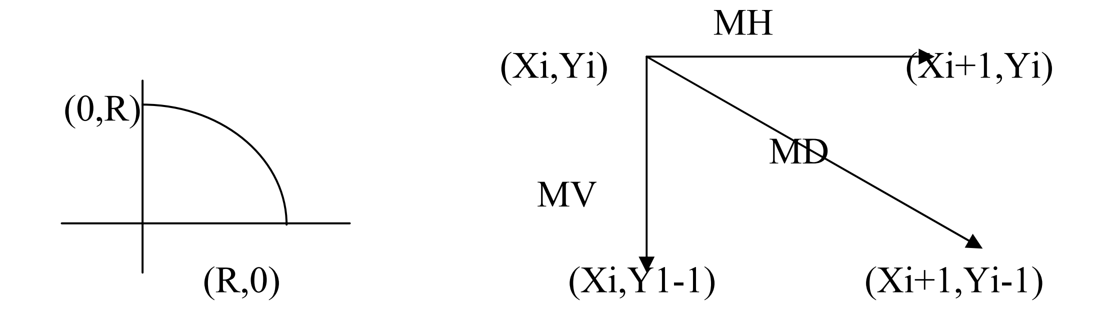
Notice that, if the algorithm begins at X=0 and Y=R (R is the radius) , then for clockwise generation of the circle Y is a monotonically decreasing function of X in the first quadrant.

Similarly if the algorithm begins at Y=0 and X=R then for counterclockwise generation of the circle, X is a decreasing function of Y.

For clockwise generation of the circle there are only three possible selections for the next pixel which best represent the circle:

1- Horizontal to the right
2- Diagonally downward to the right
3- Vertically downward

These are labeled MH , MD , MV



The algorithm chose the pixel (the movement)which minimize the square of the distance between one of these pixels and the true circle

The distance in the three cases are measured by :

Case 1 : $MH = | (X_i+1)^2 + (Y_i)^2 - R^2 |$
Case 2 : $MD = | (X_i+1)^2 + (Y_i-1)^2 - R^2 |$
Case 3 : $MV = | (X_i)^2 + (Y_i-1)^2 - R^2 |$

The difference between the square of the distance from the center of the circle to the diagonal pixel at (X_i+1 , Y_i-1) and the distance to a point on the circle R^2 is

$$D_i = (X_i+1)^2 + (Y_i-1)^2 - R^2$$

1- If $D_i<0$ then the diagonal point (X_i+1,Y_i-1) is *inside* the actual circle i.e. we use case 1 or case 2 .
   It is clear that either the pixel at (X_i+1,Y_i) == MH or that the pixel at (X_i+1,Y_i-1) == MD must be chosen.

   ə $= | (X_i+1)^2 + (Y_i)^2 - R^2| - | (X_i+1)^2 + (Y_i-1)^2 - R^2|$

   if ə <0 then the difference from the actual circle to the diagonal pixel (MD) is greater than that to the horizontal pixel (MH).

   If ə <0 choose MH (X_i+1,Y_i)
   If ə >0 choose MD (X_i+1, Y_i-1)
   The horizontal move has been selected when ə$=0$; i.e. when the distance are equal.

2- if $D_i>0$ then the diagonal point (X_i+1,Y_i-1) is *outside* the actual circle i.e. we use case 2 or case 3 .

   $\beta = | (X_i+1)^2 + (Y_i-1)^2 - R^2 | - | (X_i)^2 + (Y_i-1)^2 + R^2 |$

   if $\beta <=0$ choose MD (X_i+1, Y_i-1)
   if $\beta >0$ choose MV (X_i , Y_i-1)

3- if $D_i=0$ then we choose the pixel at (X_i+1 , Y_i-1) i.e. MD

Summery

1- Di < 0

ə ≤ 0 , then choose pixel at (Xi+1 , Yi) i.e. MH

ə > 0 , then choose pixel at (Xi+1 , Yi-1) i.e. MD

2- Di > 0

β ≤ 0 , then choose pixel at (Xi+1 , Yi-1) i.e. MD

β > 0 , then choose pixel at (Xi , Yi-1) i.e. MV

3- Di = 0

choose pixel at (Xi+1 , Yi-1) i.e. MD

Bresenham circle algorithm (for the first quadrant)

Xi=0
Yi=R
Di= 2 (1-R)
Limit = 0

1: Plot (Xi , Yi)
If Yi ≤ Limit then goto 4
If Di < 0 then goto 2
If Di > 0 then goto 3
If Di = 0 then goto 20

2: ə = 2Di + 2Yi -1
If ə ≤ 0 then goto 10
If ə > 0 then goto 20

3: β= 2Di – 2X1 -1
If β ≤ 0 then goto 20
If β > 0 then goto 30

10: Xi = Xi + 1 { MH }
Di = Di + 2Xi + 1
Goto 1

20: Xi = Xi +1 { MD }
Yi = Yi - 1
Di = Di + 2Xi – 2Yi + 2
Goto 1

30: Yi = Yi -1 { MV }
Di = Di – 2Yi + 1
Goto 1

4: Finish

Example 5 :

Draw a circle with R=4

Plot (X,Y)	Di	∂	β	Xi	Yi
	-6	-	-	0	4
(0,4)	-3	-5	-	1	4
(1,4)	-3	1	-	2	3
(2,3)	4	-1	-	3	3
(3,3)	1	-	1	3	2
(3,2)	9	-	-5	4	1
(4,1)	10	-	9	4	0
(4,0)	-3				

Computer Graphics

Two Dimension Transformation

Fundamental to all computer graphics systems is the ability to simulate the movement and the manipulation of objects in the plane. These processes are described in terms of :

1-Translation
2-Scaling
3-Rotation
4-Reflection
5-Shearing

Our object is to describe these operations in mathematical form suitable for computer processing.

There are two complementary points of view for describing object movement:

1- Geometric transformation: the object itself is moved relative to a stationary coordinate system or background. Geometric transformation is applied to each point of the object.

2- Coordinate transformation: the object is held stationary while the coordinate system is moved relative to the object, for example the motion of a car in a scene, we can keep the car fixed while moving the background scenery.

Geometric transformation

1- Translation

In translation, an object is displaced a given distance and direction from its original position. A point in the XY plane can be translated by adding translation amount to the coordinates of the point. For each point P(X,Y) which is to be moved by TX unites parallel to the X-axis and by TY units parallel to the Y-axis to the new point P2(X2,Y2) we use the equations:

$$X2=X+TX$$
$$Y2=Y+TY$$

If TX is positive then the point moves to right
If TX is negative then the point moves to left

If TY is positive then the point moves up (in PC moves down)
If TY is negative then the point moves down (in PC moves up)

The transformation of Translation can be represented by (3*3) matrix:

1	0	0
0	1	0
TX	TY	1

Example : Move the line (-4,3) , (9,-6) 3 units in the X direction and 2 units in the Y direction

Solution:

TX=3 , TY=2

First point

X1new= -4 + 3= -1
Y1new= 3 + 2= 5

Second point

X2new= 9 + 3= 12
Y2new= -6 + 2= - 4

By using the matrix representation

X1new	Y1new	1
X2new	Y2new	1

=

-4	3	1
9	-6	1

*

1	0	0
0	1	0
3	2	1

2-Scaling

Scaling is the process of expanding or compressing the dimensions of an object (changing the size of an object). The size of an object can be change by multiplying the points of an object by scaling factor.

If SF (scale factor) > 1 then the object is enlarged
If SF (scale factor) < 1 then the object is compressed
If SF (scale factor) = 1 then the object is unchanged

SX is the scale factor in the X direction
SY is the scale factor in the Y direction

To scale a point P(X,Y) we use the equations:

Xnew = X * SX
Ynew = Y * SY

If SX and SY have the same value (SX=SY) then the scaling is said to be *homogeneous* (or *balanced*).
By using the matrix representation

$$\begin{bmatrix} Xnew & Ynew & 1 \end{bmatrix} = \begin{bmatrix} X & Y & 1 \end{bmatrix} * \begin{bmatrix} SX & 0 & 0 \\ 0 & SY & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Example 1: Scale the rectangle (12,4),(20,4),(12,8),(20,8) with SX=2,SY=2

Solution : (By using the equations)

For the point (12,4)
X1new= 12 * SX =12 * 2 = 24
Y1new= 4 * SY = 4 * 2 = 8

For the point (20,4)
X2new= 20 * SX =20 * 2 = 40
Y2new= 4 * SY = 4 * 2 = 8

For the point (12,8)
X3new= 12 * SX =12 * 2 = 24
Y3new= 8 * SY = 8 * 2 = 16

For the point (20,8)
X4new= 20 * SX = 20 * 2 =40
Y4new= 8 * SY = 8 * 2 = 16

Solution : (By using matrices)

X1new	Y1new	1
X2new	Y2new	1
X3new	Y3new	1
X4new	Y4new	1

=

12	4	1
20	4	1
12	8	1
20	8	1

*

2	0	0
0	2	0
0	0	1

=

24	8	1
40	8	1
24	16	1
40	16	1

Notice that after a scaling transformation is performed, the new object is located at a different position relative to the origin. In fact, in scaling transformation the only point that remains fixed is the origin.

If we want to let one point of an object that remains at the same location (fixed), scaling can be performed by three steps:

1- Translate the fixed point to the origin, and all the points of the object must be moved the same distance and direction that the fixed point moves.

2- Scale the translated object from step one

3- Back translate the scaled object to its original position

Example 2: Scale the rectangle (12,4),(20,4),(12,8),(20,8) with SX=2,SY=2 so the point (12,4) being the fixed point.

Solution:

1- Translate the object with TX= -12 and TY= -4 so the point (12,4) lies on the origin

(12,4) ===> (0,0)
(20,4) ===> (8,0)
(12,8) ===> (0,4)
(20,8) ===> (8,4)

2- Scale the object by SX=2 and SY=2

(0,0) ===> (0 , 0)
(8,0) ===> (16 , 0)
(0,4) ===> (0 , 8)
(8,4) ===> (16 , 8)

3- Back translate the scaled object with TX= 12 and TY= 4

(0 , 0) ===> (12 , 4)
(16 , 0) ===> (28 , 4)
(0 , 8) ===> (12 ,12)
(16 , 8) ===> (28 ,12)

Example 3: Scale the triangle (80,40),(40,80),(120,80) by 0.25

Example 4: Scale the square (1,2) , (4,2) , (1,5) , (4,5) with 4 units in the X-axis ,and 2 units in the Y-axis

Solution:

X1new	Y1new	1
X2new	Y2new	1
X3new	Y3new	1
X4new	Y4new	1

=

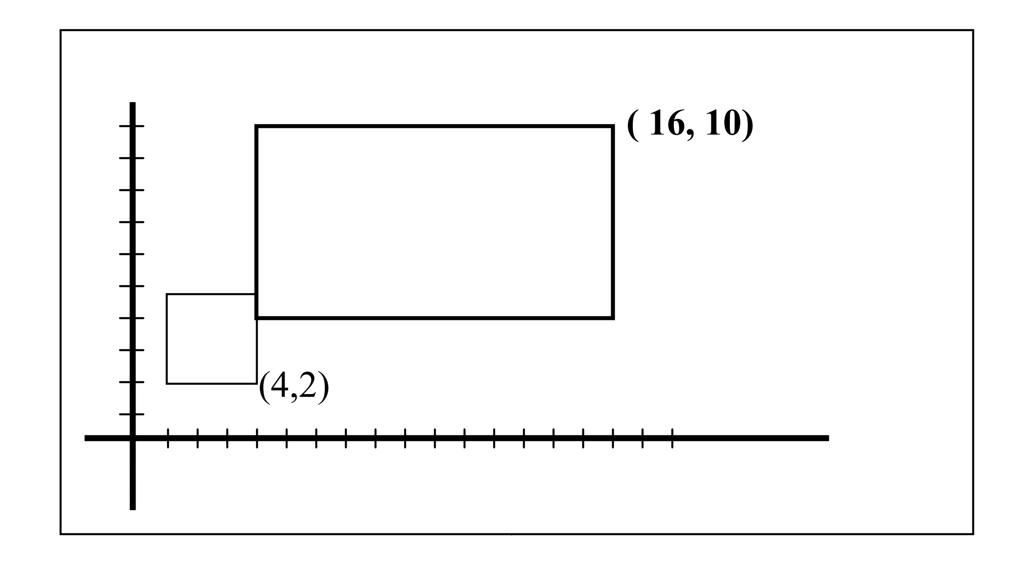
1	2	1
4	2	1
1	5	1
4	5	1

*

4	0	0
0	2	0
0	0	1

=

4	4	1
16	4	1
4	10	1
16	10	1



3-Rotation

In rotation, the object is rotated Ǿ about the origin. The convention is that the direction of rotation is counterclockwise if Ǿ is a positive angle and clockwise if Ǿ is a negative angle.

3-1 Rotation about the origin

The rotation matrix to rotate an object about the origin in anticlockwise direction is :

Cos Ǿ	Sin Ǿ	0
- Sin Ǿ	Cos Ǿ	0
0	0	1

Or, in equation :

Xnew= X * CosǾ - Y * SinǾ

Ynew= Y *Cos Ǿ + X * Sin Ǿ

When Ǿ=90, the matrix that cause a rotation through an angle of 90 (Π/2) is

0	1	0
-1	0	0
0	0	1

When Ǿ=180

-1	0	0
0	-1	0
0	0	1

When Ǿ=270

0	-1	0
1	0	0
0	0	1

When Ǿ=360

1	0	0
0	1	0
0	0	1

Example 1: rotate the line P1(1,4) and P2(3,1) anticlockwise 90 degree.
Solution:

1	4	1
3	1	1

*

0	1	0
-1	0	0
0	0	1

=

-4	1	1
-1	3	1

3-2 Rotate about a specific point (XP,YP)

We need three steps:

First: translate the points (and the object) so that the point (XP,YP) lies on the origin

XP1= X - XP
YP1= Y - YP

Second: rotate the translated point (and the translated object) by Ǿ degree about the origin to obtain the new point (XP2,YP2)

XP2=XP1 * Cos Ǿ - YP1 * Sin Ǿ
YP2=YP1 * Cos Ǿ + XP1 * Sin Ǿ

Third : Back translation
XP3= XP2 + XP
YP3= YP2 + YP

Note:: Rotation in clockwise direction :

In order to rotate in clockwise direction we use a negative angle, and because :

Cos (-Ǿ)= Cos Ǿ
Sin (-Ǿ)= - sin Ǿ

So the matrix will be (for clockwise rotation):

Cos Ǿ	-Sin Ǿ	0
Sin Ǿ	Cos Ǿ	0
0	0	1

In equations: Xnew=X * Cos Ǿ + Y * Sin Ǿ
Ynew= Y * Cos Ǿ - X * Sin Ǿ

Example 2: Rotate the square (2,1),(4,1),(2,3),(4,3) counterclockwise with Ǿ=45 around the point (2,1)

Solution : First: translate the square by TX=-2 and TY=-1

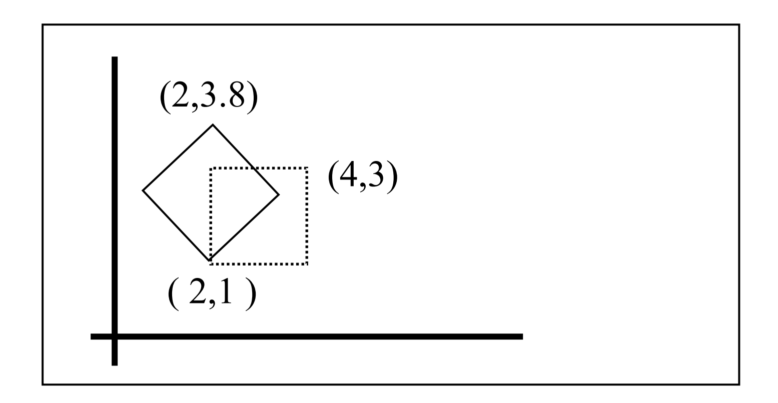
(2,1) ===> (0,0)
(4,1) ===> (2,0)
(2,3) ===> (0,2)
(4,3) ===> (2,2)

Second: Rotate by Ǿ=45

(0,0) ===> (0 , 0)
(2,0) ===> (1.414 , 1.414)
(0,2) ===> (-1.414 , 1.414)
(2,2) ===> (0 , 2.828)

Third: Back translation by TX= 2 and TY= 1

(0 , 0) ===> (2 , 1)
(1.414 , 1.414) ===> (3.414 , 2.414)
(-1.414 , 1.414) ===> (0.586 , 2.414)
(0 , 2.828) ===> (2 , 3.828)



4-Reflection

If either the X or Y axis is treated as a mirror, the object has a mirror image or reflection. The reflected point Pnew is located the same distance from the mirror (the axis) as the original point P.

4-1:Reflection on the X axis

1	0	0
0	-1	0
0	0	1

OR Xnew= X
Ynew= -Y

4-2:Reflection on the Y axis

-1	0	0
0	1	0
0	0	1

OR Xnew= - X
Ynew= Y

4-3:Reflection on the origin

-1	0	0
0	-1	0
0	0	1

OR Xnew= - X
Ynew= -Y

4-4:Reflection on the line Y=X

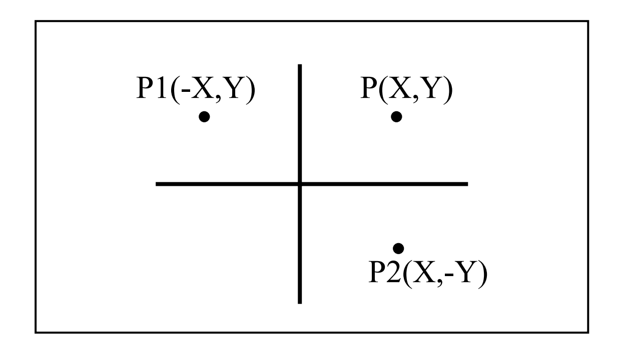
0	1	0
1	0	0
0	0	1

OR Xnew= Y
Ynew= X

4-5:Reflection on the line Y=-X

0	-1	0
-1	0	0
0	0	1

OR Xnew= -Y
Ynew= - X



Example 1: Reflect the point P(3,2) in :: a- X axis; b- Y axis; c-origin; d-line Y=X;

Solution : a-

3	2	1

*

1	0	0
0	-1	0
0	0	1

=

3	-2	1

b-

3	2	1

*

-1	0	0
0	1	0
0	0	1

=

-3	2	1

c-

3	2	1

*

-1	0	0
0	-1	0
0	0	1

=

-3	-2	1

d-

3	2	1

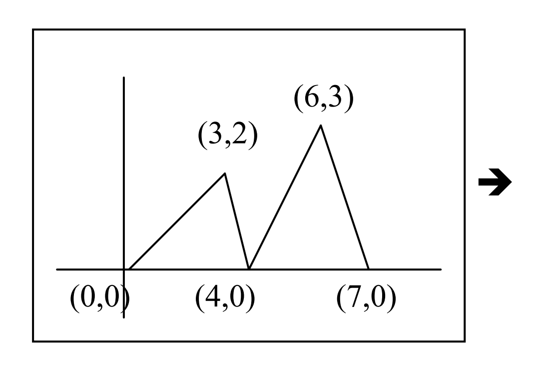
*

0	1	0
1	0	0
0	0	1

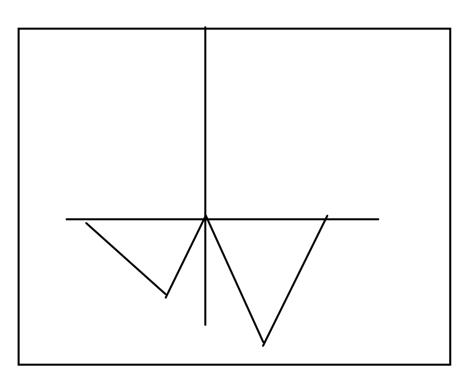
=

2	3	1

Example 2: What (3*3) matrix will change the center of the scene to the origin, and reflect the mountains in the lake? [the center of the scene is (4,0)]



➔



Solution:

First: Translate by TX= -4

1	0	0
0	1	0
-4	0	1

Second: Reflection on X axis

1	0	0
0	-1	0
0	0	1

Know multiply the two matrices :

1	0	0
0	1	0
-4	0	1

*

1	0	0
0	-1	0
0	0	1

The single matrix that perform Translation and Reflection is



1	0	0
0	-1	0
-4	0	1

0	0	1
3	2	1
4	0	1
6	3	1
7	0	1

*

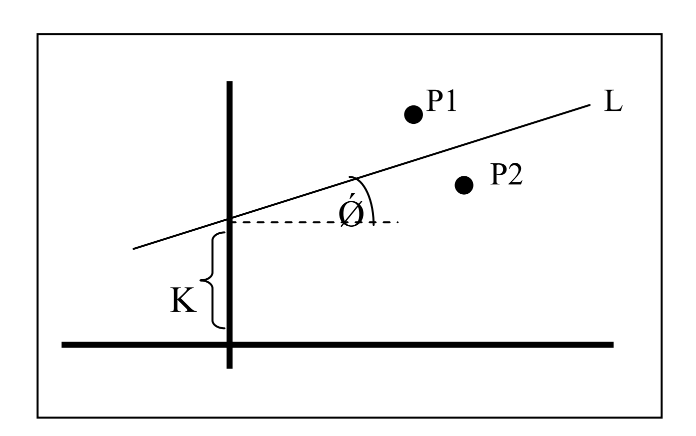
1	0	0
0	-1	0
-4	0	1

=

-4	0	1
-1	-2	1
0	0	1
2	-3	1
3	0	1

4-6:Reflection on an arbitrary line

To reflect an object on a line that does not pass through the origin, which is the general case:



As shown in the figure, let the line L intercept with Y axis in the point (0,K) and have an angle of inclination Ǿ degree with respect to the positive direction of X axis . To reflect the point P1 on the line L, we follow the following steps:

1- Move all the points up or down (in the direction of Y axis) so that L pass through the origin

T=

1	0	0
0	1	0
0	-K	1

2- Rotate all the points through (-Ǿ) degree about the origin making L lie along the X axis

R=

Cos Ǿ	-Sin Ǿ	0
Sin Ǿ	Cos Ǿ	0
0	0	1

3- Reflect the point P1 on the X axis

RefX=

1	0	0
0	-1	0
0	0	1

4- Rotate back the points by (-Ǿ) degree so that L back to its original orientation

R^{-1} =

Cos Ǿ	Sin Ǿ	0
- Sin Ǿ	Cos Ǿ	0
0	0	1

5- Shift in the direction of Y axis so that L is back in its original position

T^{-1} =

1	0	0
0	1	0
0	K	1

The sequence of matrices needed to perform this non-standard reflection is :

$S = T * R * RefX * R^{-1} * T^{-1}$

S=

Cos 2Ǿ	Sin 2Ǿ	0
Sin 2Ǿ	- Cos 2Ǿ	0
- K Sin 2Ǿ	K+K Cos 2Ǿ	1

Example 3: Find the single matrix that causes all the points in the plane to be reflected in the line with equation *Y=0.5X+2* , then apply this matrix to reflect the triangle with vertices at A(2,4), B(4,6), C(2,6) in the line.

Solution:

The Cartesian equation of a line in 2D is *Y= M * X +b* where b is the intersection of the lint with the Y axis and M is gradient of the line

$M = \Delta Y / \Delta X = Tan\ Ǿ$

So the line ***Y=0.5 X + 2*** has gradient *M= 0.5* and intersect with the Y axis at the point where *y=2*
So K=2 , Tan Ǿ =0.5 ➔ Ǿ=26.57
2Ǿ=53.13 , Cos 2Ǿ=0.6 , Sin 2Ǿ=0.8

S=

0.6	0.8	0
0.8	-0.6	0
-1.6	3.2	1

To reflect the triangle on the line :

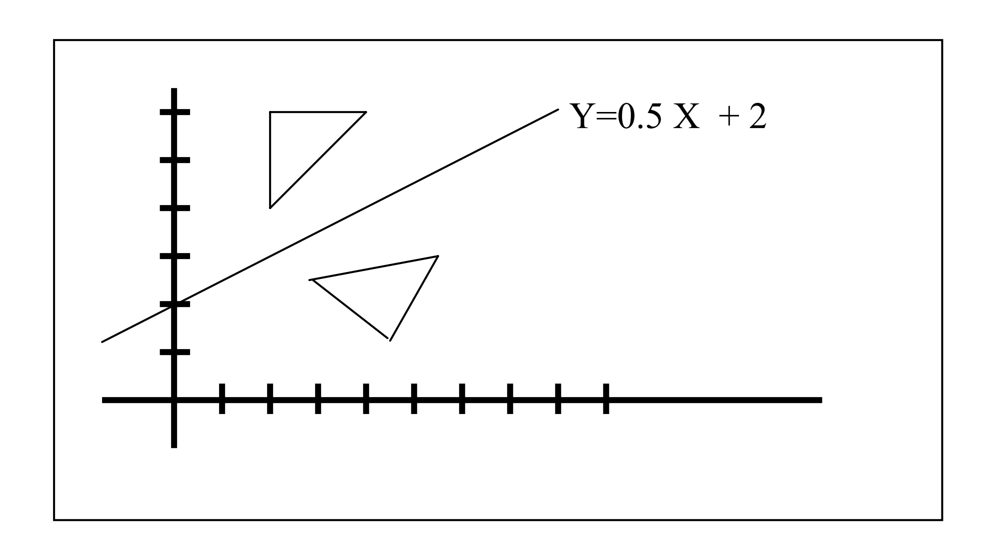
2	4	1
4	6	1
2	6	1

*

0.6	0.8	0
0.8	-0.6	0
-1.6	3.2	1

=

2.8	2.4	1
5.6	2.8	1
4.4	1.2	1



5-Shearing

It produce a distortion of an object. There are two types of shearing

1- Y shearing

It transform the point (X,Y) to the point (Xnew,Ynew) where

Xnew= X
Ynew= Y+Shy * X where $Shy \neq 0$

The matrix is

1	Shy	0
0	1	0
0	0	1

Y shearing moves a vertical line up or down depending on the sign of the shear factor Shy. A horizontal line is distorted into a line with slop Shy. And vis versa.

2- X shearing

It transform the point (X,Y) to the point (Xnew,Ynew) where

Xnew= X+Shx * Y where $Shy \neq 0$
Ynew= Y

The matrix is

1	0	0
Shx	1	0
0	0	1

Example : Share the object (1,1) , (3,1) , (1,3) , (3,3) with
a: Shx=2
b: Shy=2

Solution : a: Shx=2

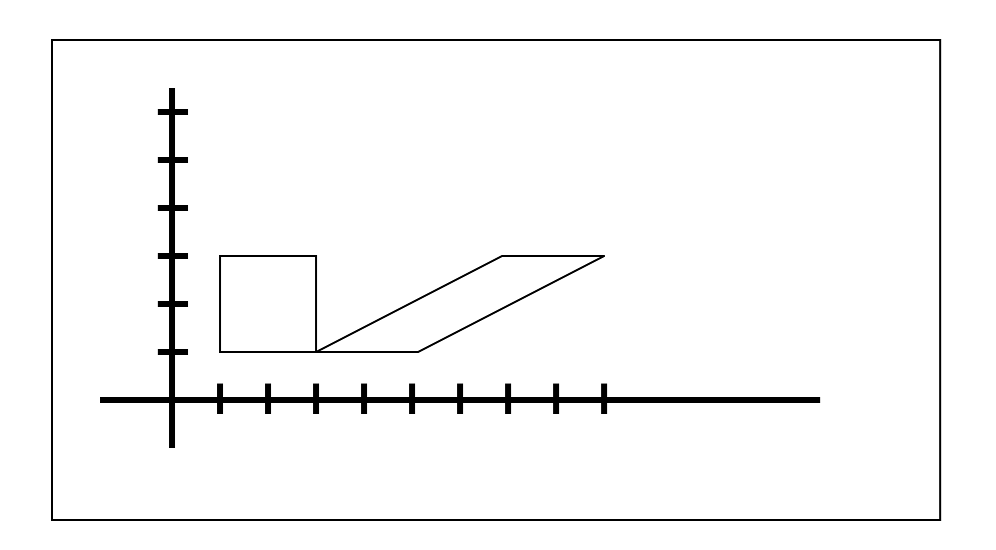
1	1	1
3	1	1
1	3	1
3	3	1

*

1	0	0
2	1	0
0	0	1

=

3	1	1
5	1	1
7	3	1
9	3	1



b- Shy

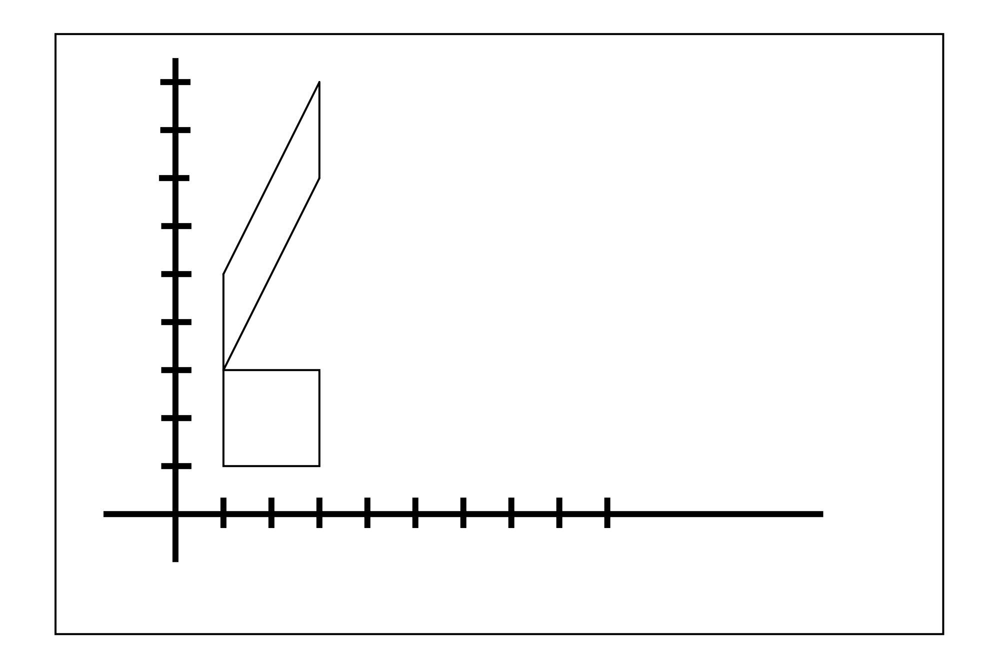
1	1	1
3	1	1
1	3	1
3	3	1

*

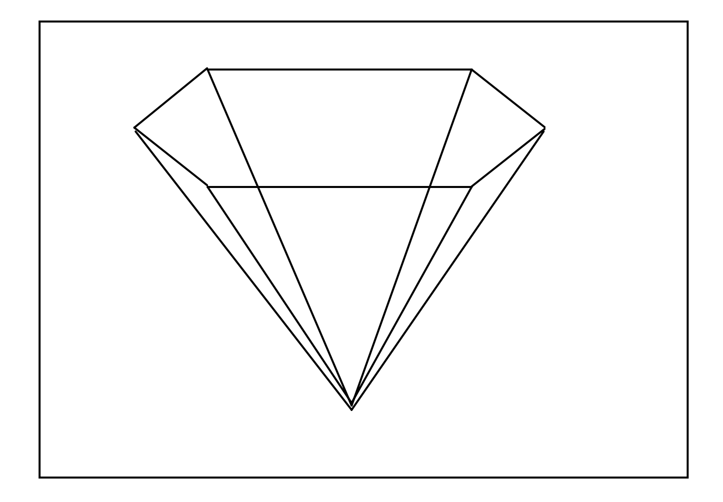
1	2	0
0	1	0
0	0	1

=

1	3	1
3	7	1
1	5	1
3	9	1



Example : Draw the object (5,30),(-5,30),(-11,24),
(-5,18),(5,18),(11,24),(0,0)



1- Share the object with Shx=-1
2- Scale on Sx=2 and Sy=1
3- Rotate the object 11 times with $\acute{\varnothing}= \Pi/6$, draw the object after each rotation

Example : Write a program in VB to generate figure 2 from figure 1

Computer Graphics

Normalized Device Coordinates

We need to use a device independent coordinate system called (NCD) to describe the viewport . This device (NCD) better than using the coordinate of the display screen to describe the viewport.

The (NCD) system allows the application programmer to write graphics programs independent of the resolution of the display screen.

World coordinate system:- is a user defined coordinate system chosen for a specific application . These screen independent coordinate can have a large or small numeric range, negative values and fraction.

Windowing :- is the capability of displaying part of the Word coordinate system image, enclosed in a rectangular region.
That is when the image described in the world coordinate system is too complicated to be viewed clearly on the screen and the user may want to view (and enlarge) only a portion of the image. Adjusting the size of the window has the effect of enlarging or shrinking or even distorting a portion of the image or the entire image.

Viewport :- it is a rectangular region of the screen where the contents of the window are displayed. That is done when we want to display different views of the image in different regions of the screen.
Using the same window and several viewport, the same object can be placed in different regions.
Choosing different windows and viewports results in a screen with multiple images.

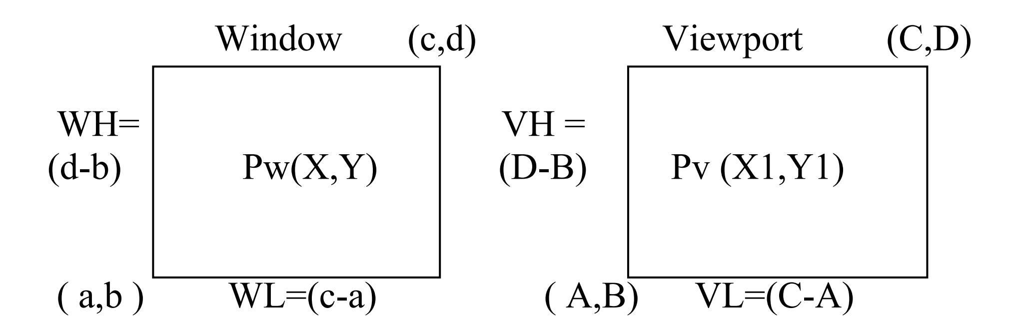
Window To Viewport mapping :

It is a transformation function that converts from Word coordinate system to (NCD). It produces on specific view of the image.
To change the view we must change the window and/or the viewport and reinvoke this mapping. The window and the viewport are described by the coordinates of the lower left vertex.

Window to viewport mapping algorithm :

We take the coordinates of the points of the object in a designated window and transform them to give the coordinates of the corresponding points in the viewport on the screen.

This mapping require three steps



1- Window Shift

Shift the lower left corner of the window to the origin:

W=

1	0	0
0	1	0
-a	-b	1

2- Scale the window dimensions to the dimensions of the viewport. The scaling involves a factor of

$$\frac{(C-A)}{(c-a)}$$ in the X direction

$$\frac{(D-B)}{(d-b)}$$ in the Y direction

The matrix of the local scaling is

$$S = \begin{bmatrix} \frac{(C-A)}{(c-a)} & 0 & 0 \\ 0 & \frac{(D-B)}{(d-b)} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3- Viewport shift

Shift the lower left corner of the viewport from the screen origin to its proper position

$$V = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ A & B & 1 \end{bmatrix}$$

Multiplying the three matrices in order, we get a single matrix for window to viewport mapping :

$$M = W * S * V$$

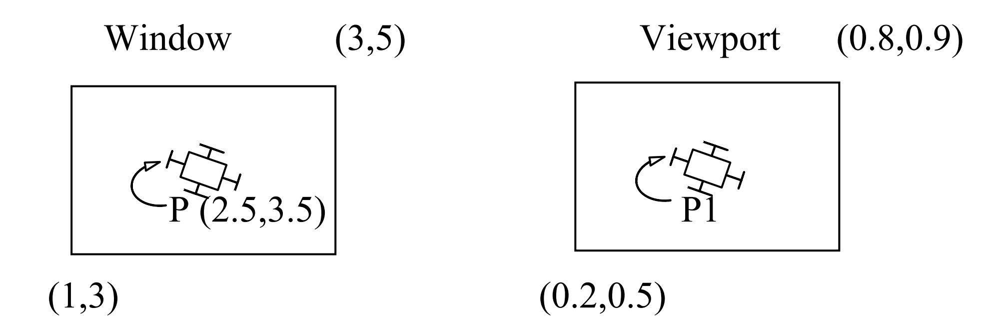
Or by using equation

$$X1 = (X - a) * \frac{VL}{WL} + A$$

$$Y1 = (Y - b) * \frac{VH}{WH} + B$$

Where (X,Y) is the point in the window and to be mapped to the point in the viewport (X1 , Y1)

Example : Point P is to be displayed on part of a screen as indicated in the figure. Describe the sequence of transformation that will transform the window shown to the viewport indicated and give the matrix for each.
Calculation the coordinates of P where it appears on the screen.



Solution

W =

1	0	0
0	1	0
-1	-3	1

S =

$\frac{0.8 - 0.2}{3 - 1}$	0	0
0	$\frac{0.9 - 0.5}{5 - 3}$	0
0	0	1

V =

1	0	0
0	1	0
0.2	0.5	1

So the single matrix M will be :

M =

0.3	0	0
0	0.2	0
-0.1	-0.1	1

P1 (in viewport) = P (inwindow) * M

= | 2.5 | 3.5 | 1 | *

0.3	0	0
0	0.2	0
-0.1	-0.1	1

= (0.65 , 0.6) in the viewport

OR by using equation

$$X1 = (2.5-1) * \frac{0.6}{2} + 0.2 = 0.65$$

$$Y1 = (3.5 - 3) * \frac{0.9 - 0.5}{5-3} + 0.5 = 0.6$$

Quiz : Generate figure 2 form figure 1 using matrices

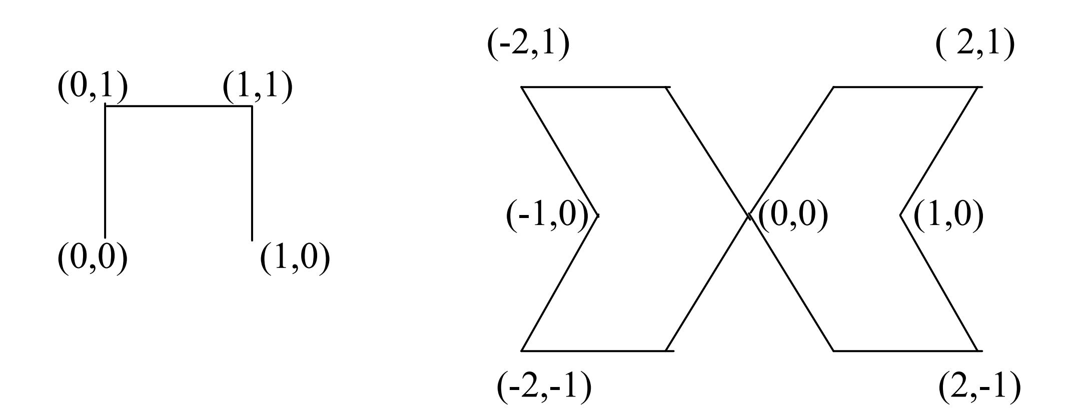


Figure 1 Figure 2

Solution

1- Shearing on X-axis by Shx=1

1	0	0
1	1	0
0	0	1

2- Reflection on X-axis

1	0	0
0	-1	0
0	0	1

3- Multiplying point matrix by (1// Shearing) then the result by (2 // Reflection)

٠	0	1
0	1	1
1	0	1
1	1	1

*

1	0	0
1	1	0
0	0	1

*

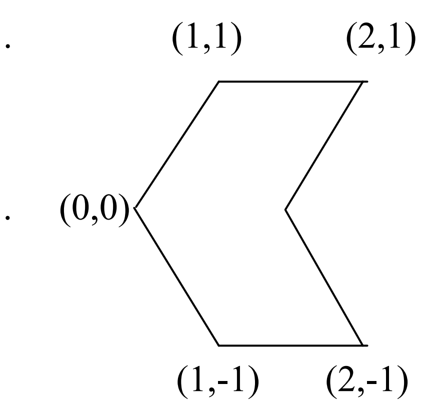
1	0	0
0	-1	0
0	0	1

=

0	0	1
1	-1	1
1	0	1
2	-1	1

Then we add the points of the new figure to the point of the origin figure and we get a figure with six points :

0	0	1
1	1	0
2	1	1
1	0	1
2	-1	1
1	-1	1



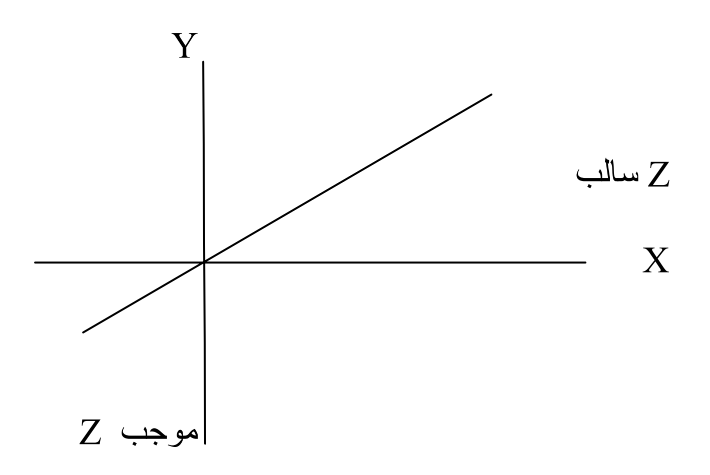
4- Reflect on Y axis

0	0	1
-1	1	0
-2	1	1
-1	0	1
-2	-1	1
-1	-1	1

Computer Graphics

Three Dimension Transformation

A point in 3D can be represented as in the figure below



1-Translation
2-Scaling
3-Rotation
4-Reflection
5-Shearing

We can perform 3D transformation by using (4 * 4) matrix for each transformation
above.

Computer Graphics

Clipping

Many graphics application programs give the user the impression of locking through a window at a very large picture. The program makes use of the scaling and translation techniques to generate a variety of different views of a single representation of a plan.

To display an enlarged portion of a picture, we must not only apply the appropriate scaling and translation but also identify the visible parts of the picture for inclusion in the displayed image.

Certain lines may lie partly inside the visible portion of the picture and partly outside.

The correct way to select visible information for display is to use clipping, a process which divides each element of the picture into its visible and invisible portions, allowing the invisible portion to be discarded.

1 Point Clipping

Is to determine if a point (X,Y) is visible or not by a simple pair of inequalities:

$$X\ left \leq X \leq X\ right$$

$$Y\ bottom \leq Y \leq Y\ top$$

Where X left , X right , Y bottom , Y top are the position of the edge of the screen.

These inequalities provides us with a very simple method of clipping pictures on a point-by-point basis.

2 Line clipping

It would be quit inappropriate to clip pictures by converting all picture elements into points and using point clipping, the clipping process would take far too long. We must instead attempt to clip large elements of the picture.

Figure (1) shows a number of different lines with respect to the screen :

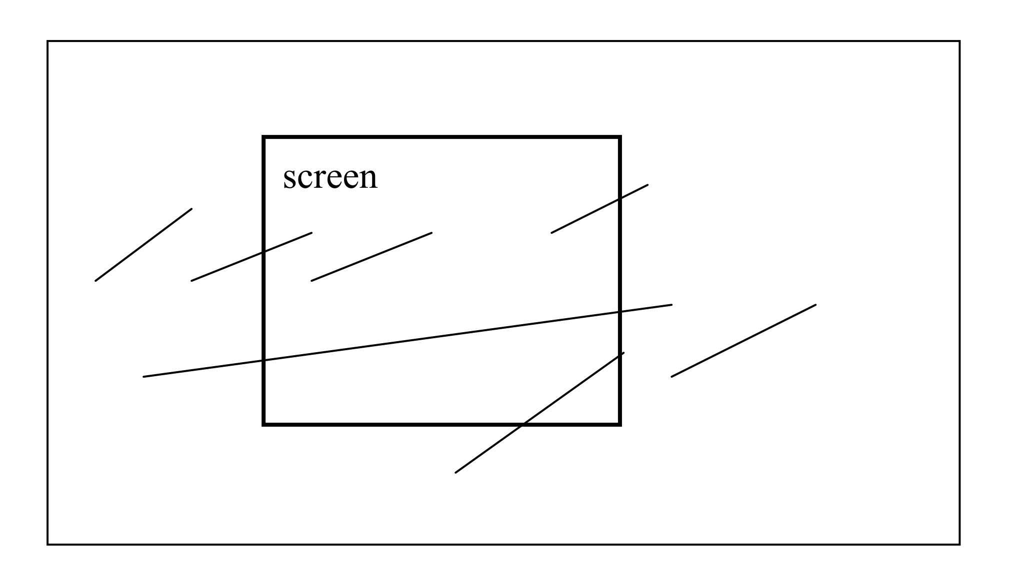


Figure – 1 –

Notice that those lines which are partly invisible are divided by the screen boundary into one or more invisible portions but into only one visible segment.

This means that the visible segment of a straight line can be determined simply by computing its two end points.

We divide the line clipping process into two phases :

1- Identify those lines which intersect the window and so need to be clipped
2- Perform the clipping

All line segments fall into one of the following clipping categories :

1- Visible : both end points of the line segment lie within the window
2- Not visible : the line segment definitely lies out side the window.

   This will occur if the line segment from (X1, Y1) to (X2 , Y2) satisfies any one of the following four inequalities :

   X1 , X2 > X max   Y1 , Y2 > Y max

   X1 , X2 < X min   Y1 , Y2 < Y min

3- Clipping candidate :the line is in neither category 1 nor category 2.

Consider figure - 2 –

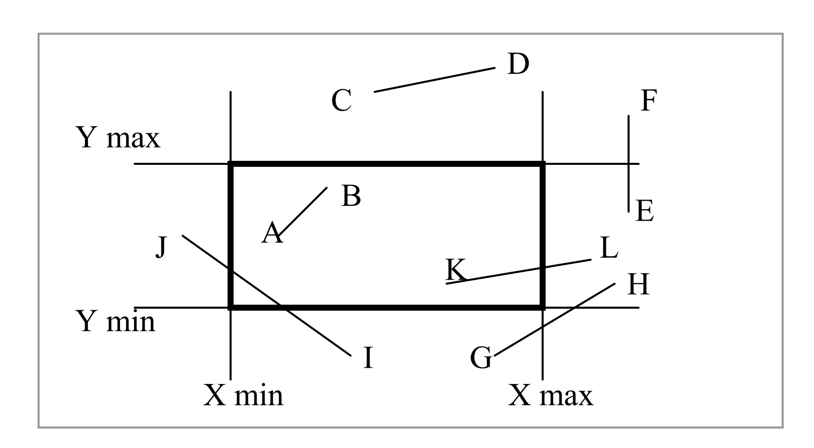


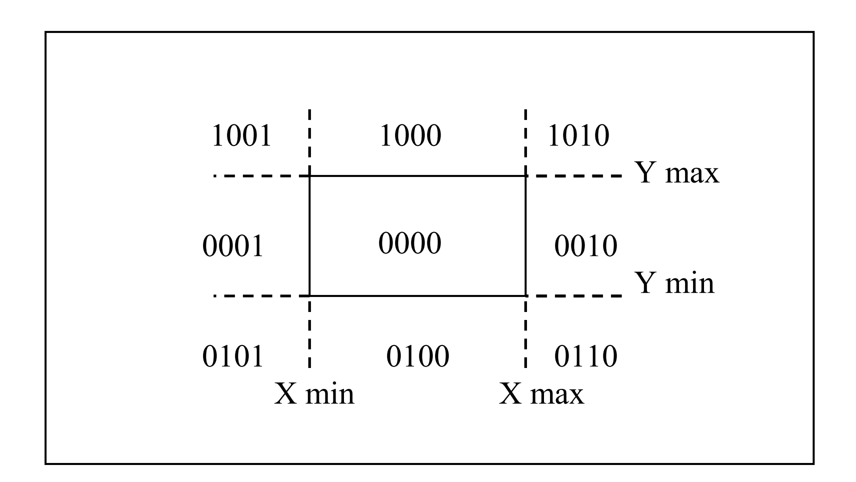
Figure - 2 -

Line segment AB is in category 1 (visible).
Line segments CD and EF are in category 2 (invisible)
Line segments GH , IJ , KL are in category 3 (clipping candidate)

Cohen – Sutherland algorithm

This algorithm is to find the category of the line segment . The algorithm proceeds in two steps :

1- Assign a 4-bit code to each endpoint of the line segments . The code is determined according to which of the following nine regions the endpoints lies in :



Starting from the leftmost bit, each bit of the code is set to true(1) of false(0) according to the schema:

Bit 1 ≡ endpoint is above the window = sign (Y – Ymax)
Bit 2 ≡ endpoint is below the window = sign (Ymin -Y)
Bit 3 ≡ endpoint is to the right of the window = sign (X - Xmax)
Bit 4 ≡ endpoint is to the left of the window = sign (Xmin - X)

Note : sign (a)=1 if a is positive , =0 otherwise

2- The line segment is visible if both endpoint codes are 0000

The line segment is not visible if the logical AND of the codes is not 0000

The line segment is candidate for clipping if the logical AND of the endpoint codes is 0000

We now decide whether the line candidate for clipping either intersect the window and so is to be clipped or don't intersect the window and so is not displayed.

The point of intersection of the line segment with an extended window edge, each of the four directions is tested in the order : left , right , top , bottom. The part of the line that is clearly outside is discarded.

To calculate the point of intersection of the line segment with the window border, the point-slope formula is used. If M is the slope of a line segment between (X1,Y1) and (X2,Y2) , then if $X1 \neq X2$:-

$$M= (Y2-Y1) / (X2-X1)$$

for any other point (X,Y) on the line :-

$$M=(Y-Y1) / (X-X1)$$

If we are testing against a left or right direction the X value is known (left or right) edge value. This X value is substituted into the equation :

$$Y=M * (X-X1) + Y1$$

If we are testing against a top or bottom, the Y value is known and substituted into :

$$X= (1/M) * (Y-Y1) + X1$$

The Cohen Algorithm

1-Determine the endpoint (X1 , Y1) code1

```
If  X1<Xmin Then
                If Y1>Ymax Then code1=1001
                If  (Y1<Ymax) And (Y1>Ymin) Then code1=0001
                If Y1<Ymin Then code1=0101
End IF

If X1>Xmax Then
                If Y1>Ymax Then code1=1010
                If  (Y1<Ymax) And (Y1>Ymin) Then code1=0010
                If Y1<Ymin Then code1=0110
End IF

If ( X1 ≤ Xmax) And ( X1 ≥ Xmin) Then
                If Y1>Ymax Then code1=1000
                If  (Y1≤Ymax) And (Y1≥Ymin) Then code1=0000
                If Y1<Ymin Then code1=0100
End IF
```

2- Determine the endpoint (X2,Y2) code2 [as code1]

3-Determine the visibility of the line

If code1= code2 = 0000 then the line is visible; draw the line

If code1 And code2 <> 0000 then the line is not visible

If code1 And code2 = 0000 then the line is candidate for clipping

4- Calculate the intersection points of the candidate lines with the window :

$$M= \Delta Y \ / \ \Delta X$$

Left : (XL,YS) : YS= M (XL-X1)+Y1 ; $M \neq \infty$
Right : (XR,YS) : YS= M (XR-X1)+Y1 ; $M \neq \infty$
Top : (XS,YT) : XS=1/M (YT-Y1)+X1 ; $M \neq 0$
Bottom : (XS,YB) : XS=1/M (YB-Y1)+X1 ; $M \neq 0$

If the candidates is out side the boundary of the window then it is Rejected.

Notes:

1- If $M=\infty$ we don't calculate the intersection points with the left edge and right edge because the line is parallel to them, we only calculate the intersection points with the top edge and bottom edge.

2- If $M=0$ we don't calculate the intersection points with the top edge and bottom edge because the line is parallel to them, we only calculate the intersection points with the left and right edge.

Example 1 : If the clipping window is XL=-4 ; XR=4 ; YT=4 ; YB=-4

Clip the lines KL=(4,5)-(6,5) and AB=(1,1)-(1,3)

Solution 1 :

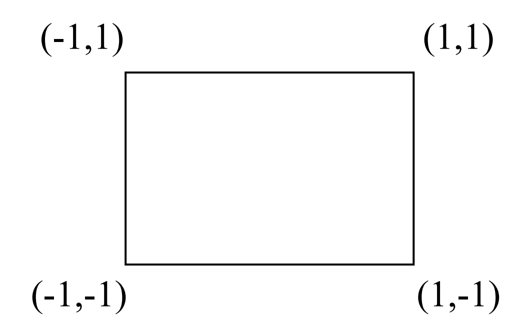
For the line KL : code1 of (4,5) is 1000
code2 of (6,5) is 1010

code1 And code2 : 1000 And 1010 = 1000 (not zero)
then the line KL is not visible

For the line AB : code1 of (1,1) is 0000
code2 of (1,3) is 0000

code1 And code2 : 0000 And 0000 = 0000 (zero)
then the line AB is visible

Example 2: Consider the clipping window



clip the line (- 3/2 , 1/6) – (1/2 , 3/2)

Solution 2 :

Code1 of (- 3/2 , 1/6) is 0001
Code2 of (1/2 , 3/2) is 1000

Code1 And Code2 : 0001 And 1000=0000
Then the line is candidate for clipping

We must calculate the intersection points:

M= 1.34 / 2 = 0.6

Left intersection : XL=-1
YL=0.6 (-1-(-3/2))+1/6=1/2
Then (-1, 0.5) is the first point of intersection

Right intersection : XR=1
YR=0.6 (1-(-3/2))+1/6=1.8
Then (1, 1.8) is rejected

Top intersection : YT=1
XT =1.6 (1- 1/6)+(- 3/2) = - 1/4
Then (-1/4,1) is the second point of intersection

Bottom intersection : YB=-1
XB =1.6 (-1 - 1/6)+(- 3/2) = - 3.25
Then (-3.25 ,-1) is rejected

3 Polygon clipping

Line clipping is acceptable when the output can be a set of disconnected lines segments. There are, however, situation in which a polygon clipped against a window should result in one or more polygons.
For example a region that is to be shaded must have a closed boundary.

The Sutherland-Hodgman polygon clipping algorithm clips a polygon against each edge of the window, one at a time. Specifically for each window edge it inputs a list of vertices and outputs a new list of vertices which is submitted to the algorithm for clipping against the next window edge.
The first window edge has as input the original set of vertices.
After clipping against the last edge, the output list consists of the vertices describing the clipped polygon.

In the algorithm, for each window edge, we tack the input list of vertices for that edge, tests a pair of consecutive vertices against a window edge to produce the output list of vertices. There are four possible cases of testing as illustrated in figure - 3 - Where testing is performed against the left edge

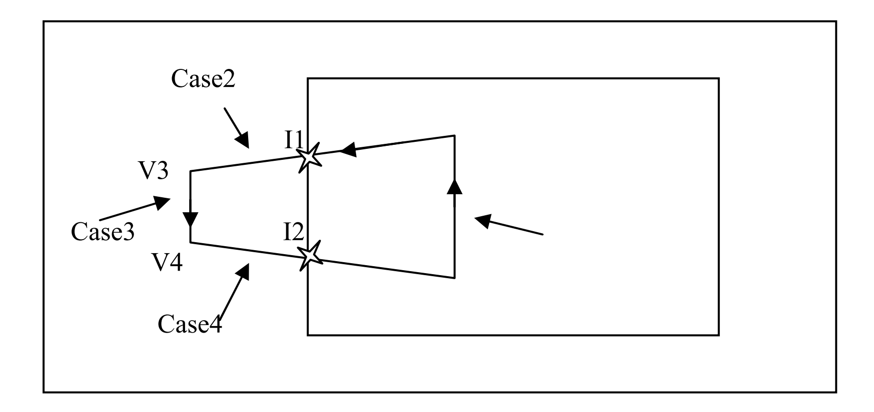


Figure - 3 -

The sample polygon has vertices { V1 , V2 , V3 , V4 } .
Case 1 has the first and second vertices V1 and V2 inside the window, so the second vertex V2 is send to the output list.

Case2 has the first vertex V2 inside and the second vertex V3 outside the window. The point of intersection, I1 , of the side of the polygon joining the vertices and the edge is added to the output list.

Case3 has both vertices, V3 and V4 outside the window and no point is output.

Case4 has the first vertex V4 outside and the second vertex V1 inside the window, the point of intersection I2 and the second vertex V1 are added to the output list.

The result of this left clipping is the transformation of the input list { V1 , V2 , V3 , V4 } to the output list { V2 , I1 , I2 , V1 }.

Programming the polygon clipping:

The declarations:

1- We need to define three arrays to hold the list of polygon vertices, each array is of two dimension, for the X and Y values of each vertex.

ORG (n , 2) / n is the number of vertices in the polygon
This matrix to store the coordinate of the vertices

POLYON (25,2) / the vertices input list {maximum 25 vertex}

POLYOUT (25,2) / the vertices output list{maximum 25 vertex }

2- We need to define a number of variables :

Vertices-no : integer ; counter of the input vertices
{same as n above}

Outlin : integer ; counter of the output vertices

Xmin,Xmax
Ymin,Ymax } Integer ; the boundary of the window

V1 (1,2) : array for the first vertex
V2 (1,2) : array for the second vertex

Edge : string ; for the edge code

3- We need two functions

V3=***Intersection*** (V1 , V2 , Edge) : used to determine the intersection point of the line between two vertices and the clip edge

Boolean=***Inside*** (V , Edge) : return true if the vertex V is inside the edge , or return false if V is outside the edge

3- the algorithm

1- Store the values of the vertices coordinate in the ORG array

2- Transfer the values in ORG to POLYIN { the initial set of vertices}

```
For I=1 to Vertices-no
  POLYIN (I,1) = ORG (I,1) { X values }
  POLYIN (I,2) = ORG (I,2) { Y values }
Next
```

3- Edge = "L" { start from the left edge of the window }

```
For  EG=1 to 4 { we have four edges in the window }
       Outlin = 0 { at first there are no output vertices }
         Take the first vertex from POLYIN and put it in V1
          For J=1 to Vertices-no
            Take the second vertex from POLYIN and put it in V2

             If Inside (V2,Edge) then  { case 1 or case 4 }
                If Inside (V1,Edge) then  { case 1}
                   Outlin=Outlin +1                 ]
                    POLYOUT (Outlin)=V2             ] output
                Else
                   V3= calculate the Intersection (V1,V2,Edge)
                    Outlin=Outlin +1                ]
                     POLYOUT (Outlin)=V3            ] output

                    Outlin=Outlin +1                ]
                    POLYOUT (Outlin)=V2             ] output
                 End IF

               Else  { case 2 or case 3 }

                If  Inside (V1, Edge) Then  { case 2 }
                   V3=Intersection (V1,V2,Edge)

                  Outlin=Outlin +1                  ]
                   POLYOUT (Outlin)=V3              ] output
                End if
               End IF

               V1=V2

          Next J
                - Set Edge to the next Edge
                - Transfer the values in POLYOUT to POLYIN
                - Vertices-no=Outlin

         Next EG
```

Example: Clip the polygon against the window

